INSTITUTO ARGENTINO DE ESTUDIOS SUPERIORES

Programación Científica

Unidad I: Generalidades

Prof. Patricia Centeno

GENERALIDADES de C++

Es un lenguaje de programación diseñado a mediados de los años 1980 por Bjarne Stroustrup. La intención de su creación fue el extender al exitoso lenguaje de programación C, desarrollado por el genial **Dennis Ritchie** entre 1969 y 1972, con mecanismos que permitan la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, el C++ es un lenguaje híbrido.

Todo programa escrito en lenguaje C++ debe:

- Estar escrito en minúsculas.
- Incluir las librerías de las funciones o comandos que necesita usar. Ejemplo #include <iostream> para poder usar cout o cin
- Comenzar con la función void main()la que debe retornar 0.
- Usar llaves {} para encerrar un bloque de sentencias. Un ejemplo de bloque es el cuerpo del programa principal de la función main().

```
int main()
{
bloque de sentencias
}
```

- Usar punto y coma (;) para finalizar cada línea de proceso.
- Solo llevan llaves las estructuras de control cuando tienen más de una instrucción o acción a realizar (exceptuando el do while), y la definición de funciones.
- Las expresiones condicionales deben ir entre paréntesis. Ejemplo: if(n<5)
- declararse using namespace std; para poder usar el cin y el cout.

TIPOS DE DATOS

TIPO	BYTES	VALOR MÍNIMO	VALOR MÁXIMO
signed char	1	-128	127
unsigned char	1	0	255
unsigned short	2	-32.768	+32.767
unsigned short	2	0	+65.535
signed int	2	-32.768	+32.767
unsigned int	2	0	+65.535
signed long	4	-2.147.483.648	+2.147.483.647
unsigned long	4	0	+4.294.967.295
float	4	3.4E-38	3.4E+38
double	8	1.7E-308	1.7E+308
long double	10	3.4E-4932	3.4E+4932
Void (vacío)	0		

Tenga en cuenta que en el lenguaje C++ **no existen funciones que conviertan una variable** de un tipo a otro, simplemente cambian de tipo de dato si se almacenan en una variable de otro tipo.

- Las variables del tipo char o short se convierten en int.
- Las variables del tipo float se convierten en doublé.
- Si alguno de los operandos es de mayor precisión que los demás, estos se convierten al tipo de aquel y el resultado es del mismo tipo.
- Si no se aplica la regla anterior y un operando es del tipo unsigned el otro se convierte en unsigned y el resultado es de este tipo.

OPERADORES

La precedencia de operadores determina el orden en que se evalúan los operadores en una expresión. Seguiremos como referencia la siguiente lista, donde los operadores de cada grupo tienen prioridad sobre los del grupo siguiente:

1. **! (not) - (menos unario) + (más unario)**
2. **operadores multiplicativos: * / %**
3. **operadores aditivos: + - (binarios)**
4. **operadores relacionales de diferencia: < <= > >=**
5. **Operadores relacionales de igualdad y desigualdad: == !=**
6. **operador lógico de conjunción: &&**
7. **operador lógico de disyunción: ||**

Además de esta lista, tenemos que tener en cuenta los siguientes puntos:

- Si dos operadores se aplican al mismo operando, el operador con más prioridad se aplica primero.
- Todos los operadores del mismo grupo tienen igual prioridad y asociatividad (se expresan de izquierda a derecha).
- Los paréntesis tienen la máxima prioridad.

Operadores Aritméticos

Símbolo	Descripción	Ejemplo	Orden de evaluación
+	SUMA	a + b	3
-	RESTA	a - b	3
*	MULTIPLICACION	a * b	2
/	DIVISION	a / b	2
%	MODULO	a % b	2
-	SIGNO	-a	1

Pre y post incremento o decremento

En el lenguaje C++, contamos con expresiones abreviadas. Por ejemplo:
int a=0; a++; es lo mismo que haber escrito a=a+1;
int b=10; b--; es lo mismo que b=b-1;

Ahora bien, resulta posible también escribir estas expresiones de la siguiente manera: *int a=0; ++a;* o *int b=10; --b;*

Escritas esas expresiones en una línea de código simple, ambas darán el mismo resultado, pero resulta lógico pensar que si existen 2 formas alguna diferencia entre ellas debe existir.

El pre – incremento/ decremento primero resuelve la operación (suma o resta 1 según el caso) y luego asigna.

El post incremento/ decremento primero asigna el valor de la variable y luego resuelve la operación (suma o resta)

El siguiente ejemplo se explica:

```
int  x;
int  y;

// Operadores de incremento
x = 1;
y = ++x;    // x es ahora 2, y es también 2
y = x++;    // x es ahora 3, y es 2

// Operadores de decremento
x = 3;
y = x--;    // x es ahora 2, y es 3
y = --x;    // x es ahora 1, y es también 1
```

Operadores Relacionales

símbolo	descripción	ejemplo	orden de evaluación
<	menor que	(a < b)	4
>	mayor que	(a > b)	4
< =	menor o igual que	(a <= b)	4
>=	mayor o igual que	(a >= b)	4
= =	igual que	(a == b)	5
! =	distinto que	(a != b)	5

Operadores Lógicos

Símbolo	Descripción	Ejemplo	Orden de evaluación
&&	Y (AND)	(a>b) && (c<d)	6
||	O (OR)	(a>b) || (c<d)	7
!	NEGACION (NOT)	!(a>b)	1

Expresiones abreviadas

Sentencia no abreviada	**Sentencia abreviada**
a = a * b	a *= b
a = a / b	a /= b
a = a % b	a %= b
a = a + b	a += b
a = a – b	a -= b

ESTRUCTURAS DE CONTROL

Selectiva o Condicional

Simple: if

```
if (condición)
    instrucción1;
```

No requiere llaves si es una única instrucción la que debe realizarse de ser verdadera la condición.

De ser más de 1 debe llevar llaves, como se ve en el siguiente ejemplo:

```
if (condición)
{
     instrucción 1;
     instrucción 2;
     instrucción 3;
}
```

Doble: if - else

```
if (condición)
    instrucción_verdadera1;
else
    instrucción_falsa1;
```

Sucede exactamente lo mismo en el caso del else, una sola instrucción no lleva llaves, 2 o más sí.

```
if (condición)
{
    instrucción_verdadera1;
    instrucción_verdadera2;
}
else
{
    instrucción_falsa1;
    instrucción_falsa2;
}
```

Múltiple: switch

La sentencia switch selecciona una de entre múltiples alternativas.
La forma general de esta expresión es la siguiente:

```
switch (expresión)
{
     case constante1:
              instrucciones;
              break;
     case constante 2:
              instrucciones;
              break;
     . . .
     default:
              instrucciones;
}
```

En una instrucción switch, expresión debe ser una expresión con un valor entero, y constante1, constante2, ..., deben ser constantes enteras, constantes de tipo carácter o una expresión constante de valor entero. Expresión también puede ser de tipo char, ya que los caracteres individuales tienen valores enteros.
Dentro de un case puede aparecer una sola instrucción o un bloque de instrucciones.
La instrucción switch evalúa la expresión entre paréntesis y compara su valor con las constantes de cada case. Se ejecutarán las instrucciones de aquel case cuya constante coincida con el valor de la expresión, y continúa hasta el final del bloque o hasta una instrucción que transfiera el control fuera del bloque del switch (una instrucción break, o return). Si no existe una constante igual al valor de la expresión, entonces se ejecutan las sentencias que están a continuación de default si existe (no es obligatorio que exista, y no tiene porqué ponerse siempre al final)

Ejemplo: Programa que lee dos números y una operación y realiza la operación entre esos números.

```
#include <iostream>
using namespace std;
int main(void)
{
    int A,B, Resultado;
    char operador;
    cout << "Introduzca un número:";
    cin >> A;
    cout << "Introduzca otro número:";
    cin >> B;
    cout <<"Introduzca un operador (+,-,*,/):";
    cin >> operador;
    Resultado = 0;
    switch (operador)
    {
           case '-' : Resultado = A - B;
                      break;
           case '+' : Resultado = A + B;
                      break;
           case '*' : Resultado = A * B;
                      break;
           case '/' : Resultado = A / B; //suponemos B!=0
                      break;
           default : cout << "Operador no valido"<< endl;
    }
    cout << "El resultado es: ";
    cout << Resultado << endl;
    return 0;
}
```

Estructuras Repetitivas

while

```
while (condición)
    instrucción 1;
```

Ejecuta una instrucción o un bloque de instrucciones cero o más veces, dependiendo del valor de la condición.

Se evalúa la condición, y si es cierta, se ejecuta la instrucción (no requiere llaves) o bloque de instrucciones y se vuelve a evaluar la condición; pero si la condición es falsa, se pasa a ejecutar la siguiente instrucción después del while.

```
while (condición)
{
     instrucción 1;
     ..............
     instrucción N;
}
```

Ejemplo: Programa que permite ingresar números enteros hasta que se ingrese un número negativo. Se muestra la suma de todos los números leídos excepto el número negativo.

```
#include <iostream>
using namespace std;
int main(void)
{
      int suma, num;
      suma = 0;
      cout << "Introduzca un numero: ";
      cin >> num;
      while (num >= 0)
      {
            suma = suma + num;
            cout << "Introduzca un numero: ";
            cin >> num;
       }
       cout << endl << "La suma es: " << suma << endl;

     return 0;
 }
```

do while

Siempre lleva llaves.

```
do {
      proposición 1 ;
} while (expresión) ;
```

```
do {
      proposición 1 ;
      proposición 2 ;
      ..............
} while (expresión) ;
```

Ejecuta una instrucción o un bloque de instrucciones, una o más veces, dependiendo del valor de la condición.

Se ejecuta la instrucción o bloque de instrucciones y a continuación se evalúa la condición. Si la condición es cierta, se vuelve a ejecutar la instrucción o bloque de instrucciones, y si es falsa, pasa a ejecutarse la siguiente instrucción después del do-while.

Cuando se utiliza una instrucción do-while el bloque de instrucciones se ejecuta al menos una vez, ya que la condición se evalúa al final. En cambio, con una instrucción while, puede suceder que el bloque de instrucciones no llegue a ejecutarse nunca si la condición inicialmente es falsa.

Ejemplo: programa que permite ingresar un número entre 1 y 100.

```
/* lee un número entre 1 y 100 */
    #include <iostream>
    using namespace std;
    int main(void)
    {
        int numero;
        do
        {
            cout << "Introduzca un numero entre 1 y 100: ";
            cin >> numero;
        }
        while (numero < 1 || numero > 100);
        return 0;
    }
```

for

Un bucle for hace que una instrucción o bloque de instrucciones se repitan un número determinado de veces mientras se cumpla la condición.

Como sucede con el if y el while, no requiere de llaves si tiene sólo una instrucción que realizar.

```
for(inicialización; condicion; incremento/decremento)
{
 instrucción 1;
 ...........
 instrucción N;
}
```

A continuación de la palabra for y entre paréntesis debe haber siempre tres zonas separadas por punto y coma:

- zona de inicialización
- zona de condición
- zona de incremento ó decremento.

En alguna ocasión puede no ser necesario escribir alguna de ellas. En ese caso se pueden dejar en blanco, pero los puntos y comas deben aparecer.

El funcionamiento de un bucle for el siguiente:

1. Se inicializa la variable o variables de control.
2. Se evalúa la condición.
3. Si la condición es cierta se ejecutan las instrucciones. Si es falsa, finaliza la ejecución del bucle y continúa el programa en la siguiente instrucción después del for
4. Se actualiza la variable o variables de control (incremento/decremento)
5. Se pasa al punto 2).

Esta instrucción es especialmente indicada para bucles donde se conozca el número de repeticiones que se van a hacer.

Como regla práctica podríamos decir que las instrucciones while y do-while se utilizan generalmente cuando no se conoce a priori el número de pasadas, y la instrucción for se utiliza generalmente cuando sí se conoce el número de pasadas.

Ejemplo: Programa que muestra los números del 1 al 10.

```
/* muestra los números de 1 a 10 */
#include <iostream>
using namespace std;
int main(void)
{
    int n;
    for (n = 1; n <= 10; n++)
    {
        cout << n << endl;
    }
return 0;
}
```